Грокаем алгоритмы. Глава 1. Знакомство с Алгоритмами

В этой главе

1. Закладываются основы для других глав
2. Вы пишите свой первый алгоритм поиска (Алгоритм бинарного поиска)
3. Вы узнаете, как описывается время выполнения алгоритма

Что такое алгоритмы?

Алгоритм - набор инструкций для выполнения некоторой задачи. В принципе любой фрагмент кода можно назвать алгоритмом, но в этой книге рассматриваются более интересные темы. В этой главе речь пойдёт о бинарном поиске.

Что вы узнаете об эффективности алгоритмов?

В каждом из алгоритмов, сначала приводится его описание и пример, а затем его время в понятиях О-большое. В завершение будут рассмотрены типы задач, которые могут решаться с применением того же алгоритма

Бинарный поиск

Предположим, вы ищите фамилию человека в телефонной книжке (жесть какая старая технология...). Она начинается с буквы "К". Конечно, мы можем листать книгу с первой странице, пока не дойдём до буквы "К", но для ускорения поиска откроем её в середине: ведь эта буква должна находится ближе к середине книги.

Или предположим, вы ищите букву "О". Тогда тоже следует начать середины.
А теперь допустим, вы придумываете username в Telegram. При этом Telegram должен проверить есть ли этот username в базе данных. Опять же поиск может начинаться с самого начала базы данных, но разумнее будет начать с середины

Перед нами типичная задача поиска. И во всех этих случаях для решения задачи можно применить один алгоритм: *бинарный поиск*

Бинарный поиск - это алгоритм; на входе он получает отсортированный список элементов. Если элемент, который вы ищите, присутствует в списке, то бинарный поиск возвращает его позицию. В противном случае null

Пример:
Сыграем в простую игру: я загадал число от 1 до 100
Вы должны отгадать мой число, использовав как можно меньше попыток. При каждой попытке я буду давать один из трёх ответов: "много", "мало", "угадал"

Предположим, вы начинаете перебирать все варианты подряд: 1, 2, 3, 4...
Вот как это будет выглядеть:
Вы: 1
Я: мало
Вы: 2
Я: мало
...
Вы: 7
Я: мало
Вы: Чёрт!

Это пример простого поиска (термин "*тупой поиск*" был бы уместнее). За раз исключается одно число. Загадай я 99, вам пришлось бы использовать 99 попыток!

Более эффективный поиск

Существует другой, более эффективный способ. Начнём с 50:
Вы: 50
Я: мало

Да, число меньше задуманного, но сейчас вы убрали *половину* чисел! Теперь вы знаете, что все число от 1 до 50 меньше загаданного. Следующая попытка:

Вы: 75
Я: много

На этот раз много, но вы снова исключили *половину* оставшихся чисел. Следующей попыткой будет 63 (число между 50 и 75):

Вы: 63
Я: много

Рано или поздно таким путём вы отгадайте загаданное число. Так и работает бинарный поиск. А теперь попробуем поточнее определить, сколько чисел будет исключаться каждый раз.

100 элементов → 50 → 25 → 13 → 7 → 4 → 2 → 1

То есть какое бы число я ни задумал, вы гарантированного его отгадайте за 7 шагов!

Предположим, что теперь чисел побольше: 240к. Как вы думаете, сколько попыток вам понадобится в худшем случае?

При простом или тупом поиске может потребоваться 240к попыток, если загадано 240000. В бинарном поиске дела обстоят намного лучше:

240к → 120к → 60к → 30к → 15к → 7.5к → … → 4 → 2 → 1

Итак, бинарный поиск потребует 18 шагов - разница заметна! В общем случае для списка из n элементов бинарный поиск выполняется за $log_2(n)$ шагов, тогда как простой поиск будет выполнен за n шагов.

Примечание
Далее в книге, когда речь пойдёт об О-большое, $O(log) == O(log_2)$

Реализация в Python

В бинарном поиске, как было показано в примерах ранее, нам надо следить с какой части массива проводится поиск. Изначально это весь массив

```python
low = 0
hight = len(lst) - 1
```

Каждый раз алгоритм должен проверять средний элемент:

```python
mid = (low + high) // 2
guess = lst[mid]
```

Если названое число было мало, то переменная low обновляется:

```python
if guess < item:
        low = mid + 1
```

Иначе обновляется переменная high
Полный код выглядит так:

```python
def binary_search(lst, item):
        low = 0
        high = len(lst) - 1
        while low <= high:
                mid = (low + high) // 2
                guess = lst[mid]
                if guess == item:
                        return mid
                elif guess > item:
                        high = mid - 1
                else:
                        low = mid + 1
        return None
```

Время выполнения

Каждый раз, когда мы будем рассматривать очередной алгоритм, я буду обсуждать время его выполнения. Обычно следует выбирать самый эффективный алгоритм, будь то оптимизация по времени или памяти

Если вернуться к сравнению простого поиска и бинарного, то время первого обычно называют *линейным* ($O(n)$ - если говорить в терминах О-большого), ибо в худшем случае он всегда выполняется за n попыток при n

элементов. Время же второго называется *логарифмическим* ($O(log(n))$) по, наверное, понятным причинам.

О-большое

Специальная нотация "О-большое" описывает скорость работы разных алгоритмов. Зачем вам это? Время от времени вам придётся использовать чужие алгоритмы, а потому неплохо было бы понимать, насколько быстро или медленно они работают. О-большое описывает, насколько быстро работает алгоритм, оно НЕ сообщает скорость в секундах, а *позволяет сравнить количество операций*. Оно указывает, насколько быстро возрастает время выполнения алгоритма.

ВАЖНО!
О-большое описывает количество операций в ХУДШЕМ случае. Если простой поиск нашёл человека в телефонной книжке за одну операцию, потому что он был первым, это ничего не значит! Скорость выполнения простого поиска всё равно $O(n)$.

Примечание
Помимо худшего времени также полезно учитывать среднее время выполнения

Типичные примеры "О-большое"

- $O(log(n))$ - логарифмическое время. Пример: бинарный поиск
- $O(n)$ - линейное время. Пример: простой поиск
- $O(n \cdot log(n))$. Пример: эффективные алгоритмы сортировки
- $O(n^2)$. Пример: медленные алгоритмы сортировки
- $O(n!)$. Пример: очень медленные алгоритмы

Итоги

- Скорость алгоритмов измеряется не в секундах, а в темпе роста количества операций
- По сути, формула описывает, насколько быстро возрастает время выполнения алгоритма с увеличением размера входных данных.
- Время выполнения алгоритмов выражается как "О-большое"

Упражнения к главе

(Оставлю их здесь для тех, кому это интересно. Напишите в комментарии или мне, если нужны будут ответы)

1. Имеется отсортированный список из 128 имён, и вы ищите в нём значение методом бинарного поиска. Какое максимально число проверок для этого может потребоваться?
2. Предположим, список увеличился вдвое. Как изменится максимальное количество проверок?

Приведите время выполнения "О-большое" для каждого из сценариев:

3. Известна фамилия, нужно найти номер в телефонной книге.
4. Известен номер, нужно найти фамилию в телефонной книге.
5. Нужно прочитать телефоны всех людей в телефонной книге.
6. Нужно прочитать телефоны всех людей, фамилии которых начинаются на "А" (Вопрос с подвохом!)